МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО"

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Лабораторна робота №2

з дисципліни «Бази даних»

тема «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав(ла)	Перевірив
студент(ка) II курсу	"____" "____________" 20___ р.
групи КП-03	викладач
Сітайло Анна Сергіївна	Радченко Костянтин
(*прізвище, ім'я, по батькові*)	Олександрович
	(*прізвище, ім'я, по батькові*)

Варіант №17

Київ 2021

Мета

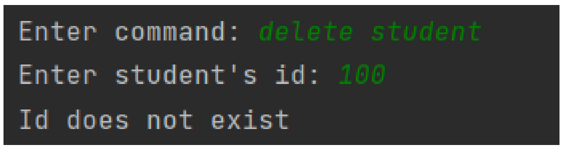
Здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Завдання

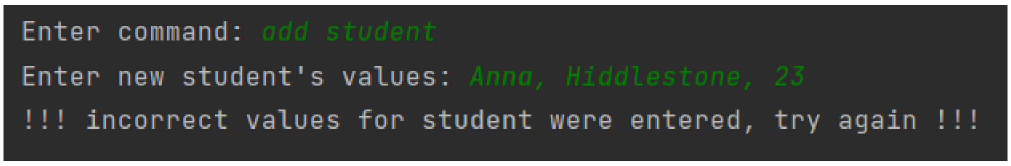
1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Результати

1. Вимоги до 1-го пункту деталізованого завдання

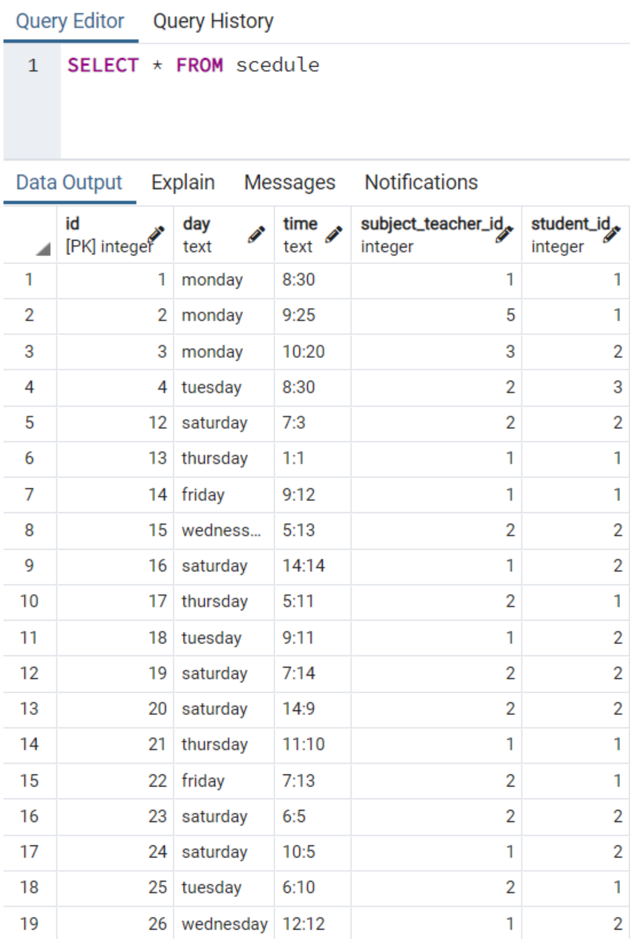
a. 

```
Enter command: delete student
Enter student's id: 100
Id does not exist
```

b. 

```
Enter command: add student
Enter new student's values: Anna, Hiddlestone, 23
!!! incorrect values for student were entered, try again !!!
```

2. Вимоги до 2-го пункту деталізованого завдання

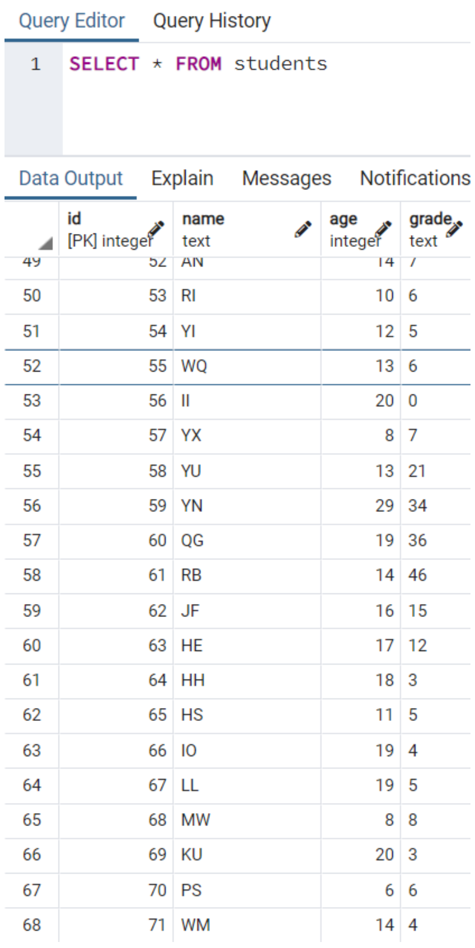


Query Editor Query History

```sql
SELECT * FROM scedule
```

Data Output Explain Messages Notifications

	id [PK] integer	day text	time text	subject_teacher_id integer	student_id integer
1	1	monday	8:30	1	1
2	2	monday	9:25	5	1
3	3	monday	10:20	3	2
4	4	tuesday	8:30	2	3
5	12	saturday	7:3	2	2
6	13	thursday	1:1	1	1
7	14	friday	9:12	1	1
8	15	wedness...	5:13	2	2
9	16	saturday	14:14	1	2
10	17	thursday	5:11	2	1
11	18	tuesday	9:11	1	2
12	19	saturday	7:14	2	2
13	20	saturday	14:9	2	2
14	21	thursday	11:10	1	1
15	22	friday	7:13	2	1
16	23	saturday	6:5	2	2
17	24	saturday	10:5	1	2
18	25	tuesday	6:10	2	1
19	26	wednesday	12:12	1	2

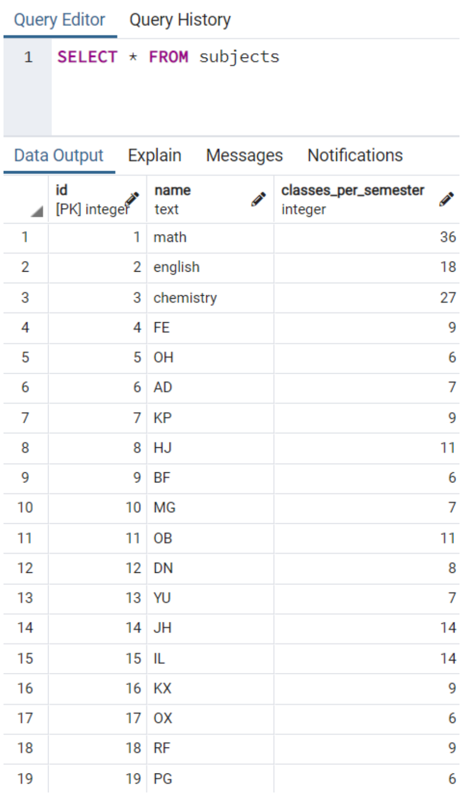


Query Editor Query History

```sql
SELECT * FROM students
```

Data Output Explain Messages Notifications

	id [PK] integer	name text	age integer	grade text
49	52	AN	14	7
50	53	RI	10	6
51	54	YI	12	5
52	55	WQ	13	6
53	56	II	20	0
54	57	YX	8	7
55	58	YU	13	21
56	59	YN	29	34
57	60	QG	19	36
58	61	RB	14	46
59	62	JF	16	15
60	63	HE	17	12
61	64	HH	18	3
62	65	HS	11	5
63	66	IO	19	4
64	67	LL	19	5
65	68	MW	8	8
66	69	KU	20	3
67	70	PS	6	6
68	71	WM	14	4

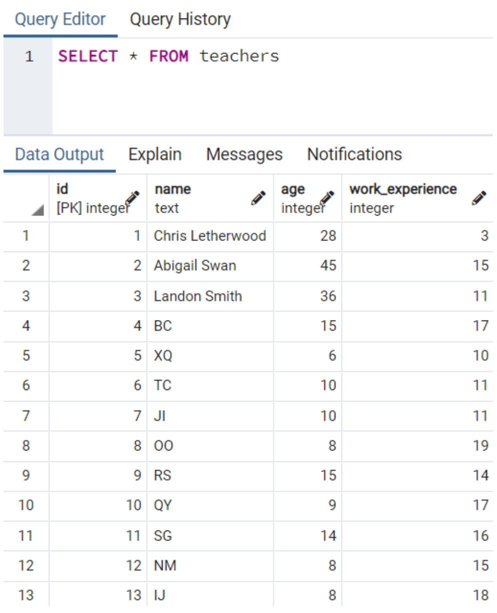


Query Editor Query History

```sql
SELECT * FROM subjects
```

Data Output Explain Messages Notifications

	id [PK] integer	name text	classes_per_semester integer
1	1	math	36
2	2	english	18
3	3	chemistry	27
4	4	FE	9
5	5	OH	6
6	6	AD	7
7	7	KP	9
8	8	HJ	11
9	9	BF	6
10	10	MG	7
11	11	OB	11
12	12	DN	8
13	13	YU	7
14	14	JH	14
15	15	IL	14
16	16	KX	9
17	17	OX	6
18	18	RF	9
19	19	PG	6

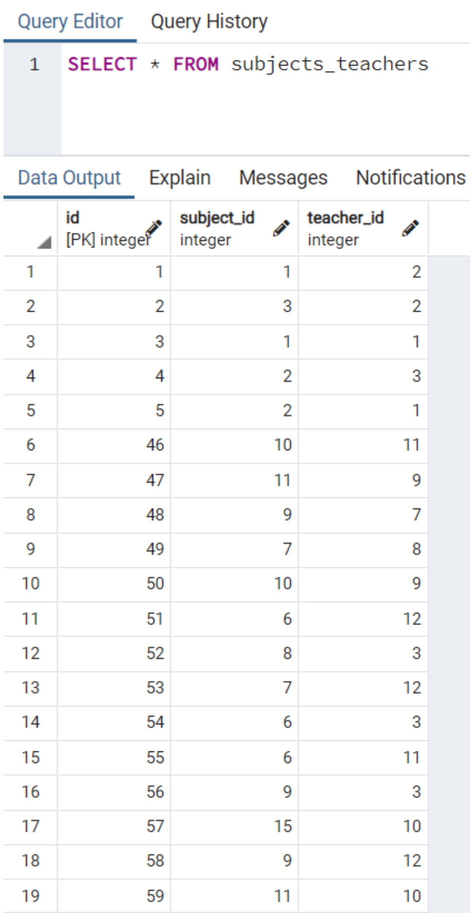


Query Editor Query History

```sql
SELECT * FROM teachers
```

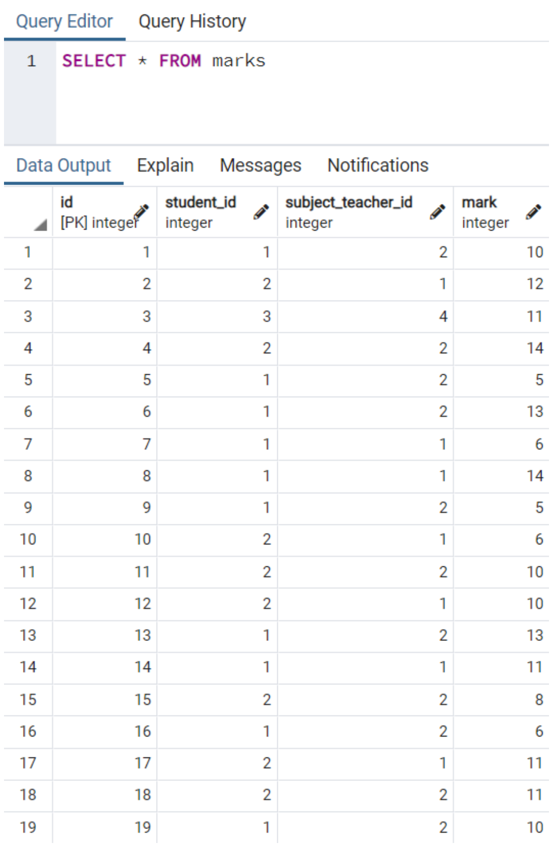
Data Output Explain Messages Notifications

	id [PK] integer	name text	age integer	work_experience integer
1	1	Chris Letherwood	28	3
2	2	Abigail Swan	45	15
3	3	Landon Smith	36	11
4	4	BC	15	17
5	5	XQ	6	10
6	6	TC	10	11
7	7	JI	10	11
8	8	OO	8	19
9	9	RS	15	14
10	10	QY	9	17
11	11	SG	14	16
12	12	NM	8	15
13	13	IJ	8	18



```sql
SELECT * FROM subjects_teachers
```

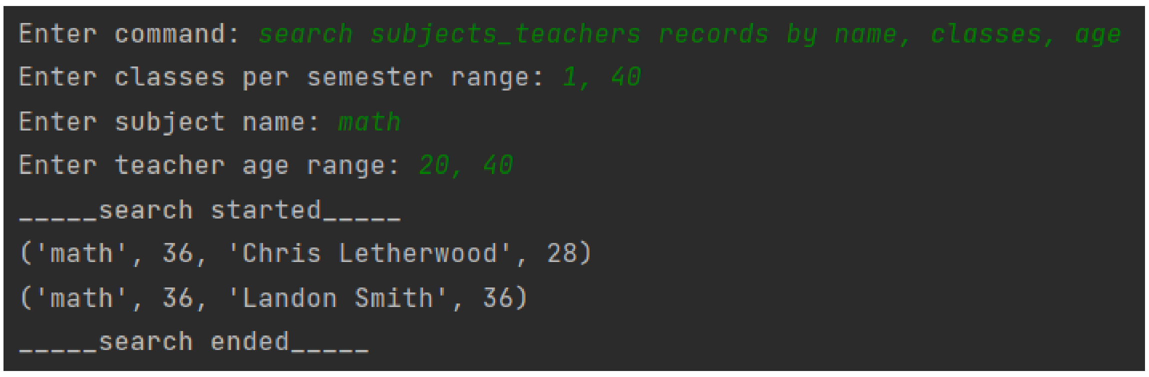
	id [PK] integer	subject_id integer	teacher_id integer
1	1	1	2
2	2	3	2
3	3	1	1
4	4	2	3
5	5	2	1
6	46	10	11
7	47	11	9
8	48	9	7
9	49	7	8
10	50	10	9
11	51	6	12
12	52	8	3
13	53	7	12
14	54	6	3
15	55	6	11
16	56	9	3
17	57	15	10
18	58	9	12
19	59	11	10



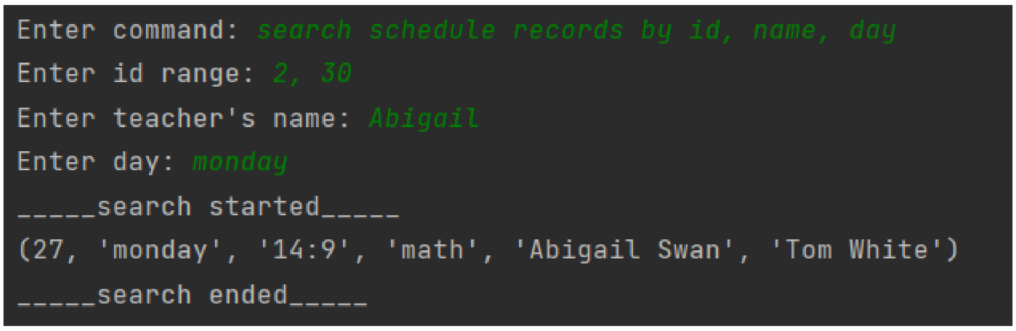
```sql
SELECT * FROM marks
```

	id [PK] integer	student_id integer	subject_teacher_id integer	mark integer
1	1	1	2	10
2	2	2	1	12
3	3	3	4	11
4	4	2	2	14
5	5	1	2	5
6	6	1	2	13
7	7	1	1	6
8	8	1	1	14
9	9	1	2	5
10	10	2	1	6
11	11	2	2	10
12	12	2	1	10
13	13	1	2	13
14	14	1	1	11
15	15	2	2	8
16	16	1	2	6
17	17	2	1	11
18	18	2	2	11
19	19	1	2	10

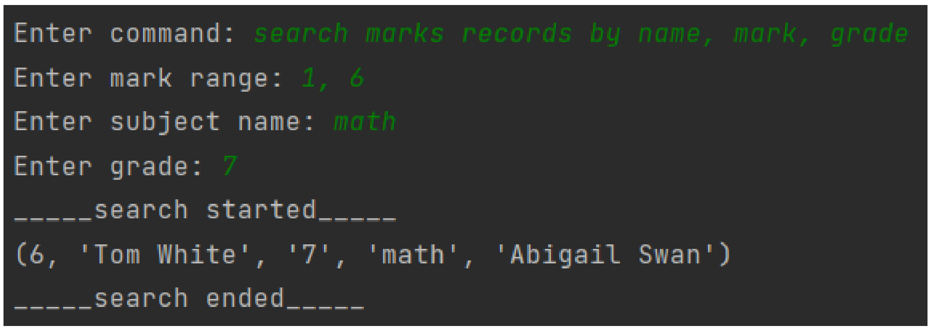
3. Вимоги до 3-го пункту деталізованого завдання



```
Enter command: search subjects_teachers records by name, classes, age
Enter classes per semester range: 1, 40
Enter subject name: math
Enter teacher age range: 20, 40
_____search started_____
('math', 36, 'Chris Letherwood', 28)
('math', 36, 'Landon Smith', 36)
_____search ended_____
```

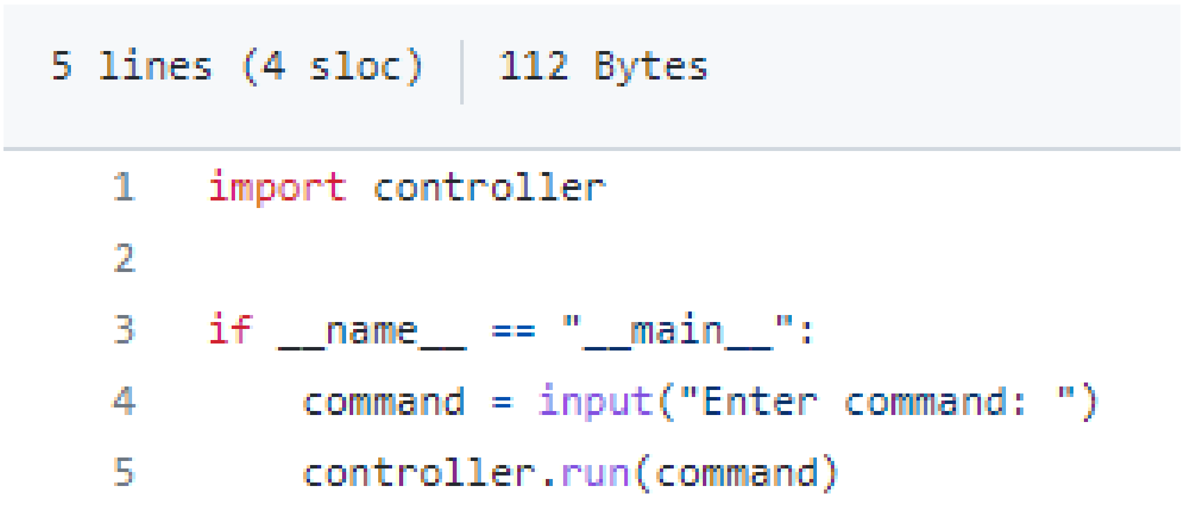


```
Enter command: search schedule records by id, name, day
Enter id range: 2, 30
Enter teacher's name: Abigail
Enter day: monday
_____search started_____
(27, 'monday', '14:9', 'math', 'Abigail Swan', 'Tom White')
_____search ended_____
```



```
Enter command: search marks records by name, mark, grade
Enter mark range: 1, 6
Enter subject name: math
Enter grade: 7
_____search started_____
(6, 'Tom White', '7', 'math', 'Abigail Swan')
_____search ended_____
```

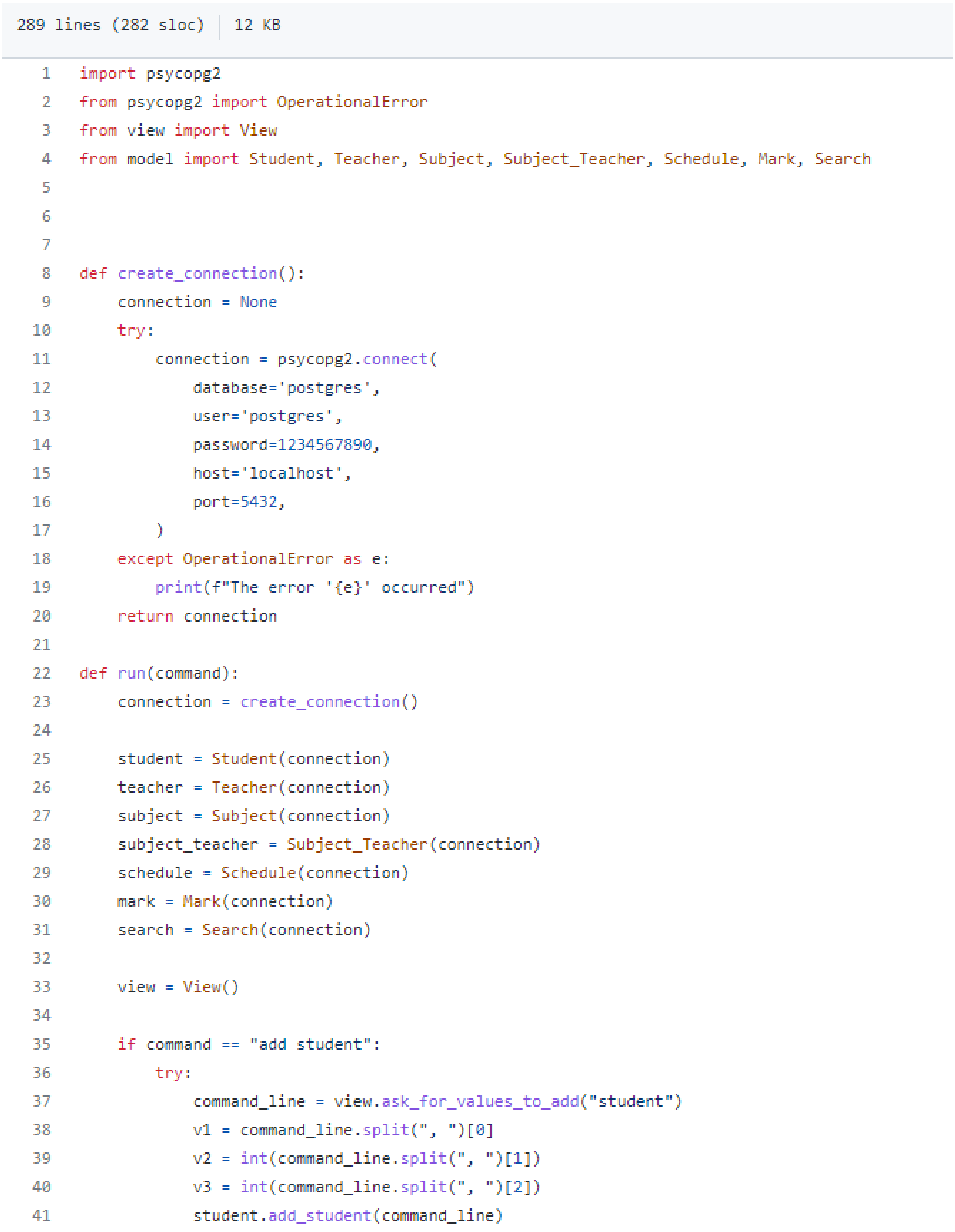
4. Вимоги до 4-го пункту деталізованого завдання
Main.py:



```
5 lines (4 sloc) | 112 Bytes
```

```python
import controller

if __name__ == "__main__":
    command = input("Enter command: ")
    controller.run(command)
```

Controller.py:



```
289 lines (282 sloc) | 12 KB
```

```python
import psycopg2
from psycopg2 import OperationalError
from view import View
from model import Student, Teacher, Subject, Subject_Teacher, Schedule, Mark, Search



def create_connection():
    connection = None
    try:
        connection = psycopg2.connect(
            database='postgres',
            user='postgres',
            password=1234567890,
            host='localhost',
            port=5432,
        )
    except OperationalError as e:
        print(f"The error '{e}' occurred")
    return connection

def run(command):
    connection = create_connection()

    student = Student(connection)
    teacher = Teacher(connection)
    subject = Subject(connection)
    subject_teacher = Subject_Teacher(connection)
    schedule = Schedule(connection)
    mark = Mark(connection)
    search = Search(connection)

    view = View()

    if command == "add student":
        try:
            command_line = view.ask_for_values_to_add("student")
            v1 = command_line.split(", ")[0]
            v2 = int(command_line.split(", ")[1])
            v3 = int(command_line.split(", ")[2])
            student.add_student(command_line)
```

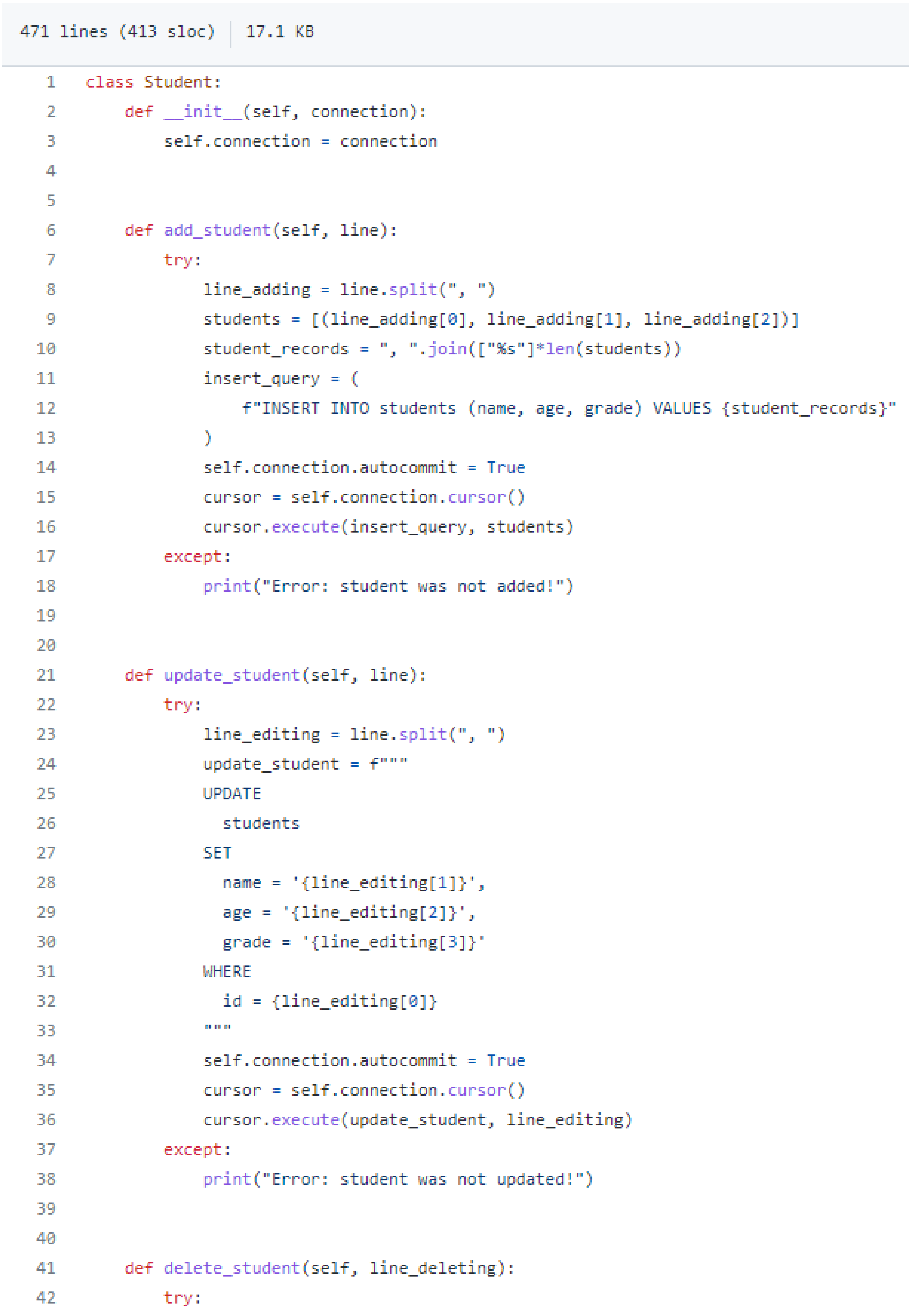
```python
                command_line = view.ask_for_values_to_add("student")
                v1 = command_line.split(", ")[0]
                v2 = int(command_line.split(", ")[1])
                v3 = int(command_line.split(", ")[2])
                student.add_student(command_line)
                view.added_message("student")
            except:
                view.incorrect_input_message("student")
        elif command == "update student":
            try:
                command_line = view.ask_for_values_to_update("student")
                v1 = command_line.split(", ")[0]
                v2 = int(command_line.split(", ")[1])
                v3 = int(command_line.split(", ")[2])
                student.update_student(command_line)
                view.updated_message("student")
            except:
                view.incorrect_input_message("student")
        elif command == "delete student":
            try:
                command_line = int(view.ask_for_values_to_delete("student"))
                student.delete_student(command_line)
                view.deleted_message("student")
            except:
                view.incorrect_input_message("student")
        elif command == "get random students":
            try:
                command_line = int(view.ask_for_values_to_generate("student"))
                student.generate_random_students(command_line)
                view.generated_message("student")
            except:
                view.incorrect_input_message("student")
        elif command == "add teacher":
            try:
                command_line = view.ask_for_values_to_add("teacher")
                v1 = command_line.split(", ")[0]
                v2 = int(command_line.split(", ")[1])
                v3 = int(command_line.split(", ")[2])
                teacher.add_teacher(command_line)
                view.added_message("teacher")
            except:
                view.incorrect_input_message("teacher")
        elif command == "update teacher":
            try:
                command_line = view.ask_for_values_to_update("teacher")
                command_line = view.ask_for_values_to_update("teacher")
                v1 = command_line.split(", ")[0]
                v2 = int(command_line.split(", ")[1])
                v3 = int(command_line.split(", ")[2])
                teacher.update_teacher(command_line)
                view.updated_message("teacher")
            except:
                view.incorrect_input_message("teacher")
        elif command == "delete teacher":
            try:
                command_line = int(view.ask_for_values_to_delete("teacher"))
                teacher.delete_teacher(command_line)
                view.deleted_message("teacher")
            except:
                view.incorrect_input_message("teacher")
        elif command == "get random teachers":
            try:
                command_line = int(view.ask_for_values_to_generate("teacher"))
                teacher.generate_random_teachers(command_line)
                view.generated_message("teacher")
            except:
                view.incorrect_input_message("teacher")
        elif command == "add subject":
            try:
                command_line = view.ask_for_values_to_add("subject")
                v1 = command_line.split(", ")[0]
                v2 = int(command_line.split(", ")[1])
                subject.add_subject(command_line)
                view.added_message("subject")
            except:
                view.incorrect_input_message("subject")
        elif command == "update subject":
            try:
                command_line = view.ask_for_values_to_update("subject")
                v1 = command_line.split(", ")[0]
                v2 = int(command_line.split(", ")[1])
                subject.update_subject(command_line)
                view.updated_message("subject")
            except:
                view.incorrect_input_message("subject")
        elif command == "delete subject":
            try:
                command_line = view.ask_for_values_to_delete("subject")
                v = int(command_line)
                subject.delete_subject(command_line)
```

```python
                subject.delete_subject(command_line)
                view.deleted_message("subject")
            except:
                view.incorrect_input_message("subject")
        elif command == "get random subjects":
            try:
                command_line = view.ask_for_values_to_generate("subject")
                v = int(command_line)
                subject.generate_random_subjects(command_line)
                view.generated_message("subject")
            except:
                view.incorrect_input_message("subject")
        elif command == "add subjects_teachers record":
            try:
                command_line = view.ask_for_values_to_add("subjects_teachers record")
                v1 = int(command_line.split(", ")[0])
                v2 = int(command_line.split(", ")[1])
                subject_teacher.add_subjects_teachers_record(command_line)
                view.added_message("subjects_teachers record")
            except:
                view.incorrect_input_message("subjects_teachers record")
        elif command == "update subjects_teachers record":
            try:
                command_line = view.ask_for_values_to_update("subjects_teachers record")
                v1 = int(command_line.split(", ")[0])
                v2 = int(command_line.split(", ")[1])
                subject_teacher.update_subjects_teachers_record(command_line)
                view.updated_message("subjects_teachers record")
            except:
                view.incorrect_input_message("subjects_teachers record")
        elif command == "delete subjects_teachers record":
            try:
                command_line = view.ask_for_values_to_delete("subjects_teachers record")
                v = int(command_line)
                subject_teacher.delete_subjects_teachers_record(command_line)
                view.deleted_message("subjects_teachers record")
            except:
                view.incorrect_input_message("subjects_teachers record")
        elif command == "get random subjects_teachers records":
            try:
                command_line = view.ask_for_values_to_generate("subjects_teachers record")
                v = int(command_line)
                subject_teacher.generate_random_subjects_teachers_records(command_line)
                view.generated_message("subjects_teachers record")
            except:
```

```python
            except:
                view.incorrect_input_message("subjects_teachers record")
        elif command == "add schedule record":
            try:
                command_line = view.ask_for_values_to_add("schedule")
                v1 = command_line.split(", ")[0]
                v2 = command_line.split(", ")[1]
                v3 = int(command_line.split(", ")[2])
                v4 = int(command_line.split(", ")[3])
                schedule.add_schedule_record(command_line)
                view.added_message("schedule")
            except:
                view.incorrect_input_message("schedule")
        elif command == "update schedule record":
            try:
                command_line = view.ask_for_values_to_update("schedule")
                v1 = command_line.split(", ")[0]
                v2 = command_line.split(", ")[1]
                v3 = int(command_line.split(", ")[2])
                v4 = int(command_line.split(", ")[3])
                schedule.update_schedule_record(command_line)
                view.updated_message("schedule")
            except:
                view.incorrect_input_message("schedule")
        elif command == "delete schedule record":
            try:
                command_line = view.ask_for_values_to_delete("schedule")
                v = int(command_line)
                schedule.delete_schedule_record(command_line)
                view.deleted_message("schedule")
            except:
                view.incorrect_input_message("schedule")
        elif command == "get random schedule records":
            try:
                command_line = view.ask_for_values_to_generate("schedule")
                v = int(command_line)
                schedule.generate_random_schedule_records(command_line)
                view.generated_message("schedule")
            except:
                view.incorrect_input_message("schedule")
        elif command == "add mark record":
            try:
                command_line = view.ask_for_values_to_add("mark record")
                v1 = int(command_line.split(", ")[0])
                v2 = int(command_line.split(", ")[1])
                v2 = int(command_line.split(", ")[1])
                v3 = int(command_line.split(", ")[2])
                mark.add_mark(command_line)
                view.added_message("mark record")
            except:
                view.incorrect_input_message("mark record")
        elif command == "update mark record":
            try:
                command_line = view.ask_for_values_to_update("mark record")
                v1 = int(command_line.split(", ")[0])
                v2 = int(command_line.split(", ")[1])
                v3 = int(command_line.split(", ")[2])
                mark.update_mark(command_line)
                view.updated_message("mark record")
            except:
                view.incorrect_input_message("mark record")
        elif command == "delete mark record":
            try:
                command_line = view.ask_for_values_to_delete("mark record")
                v = int(command_line)
                mark.delete_mark(command_line)
                view.deleted_message("mark record")
            except:
                view.incorrect_input_message("mark record")
        elif command == "get random mark records":
            try:
                command_line = view.ask_for_values_to_generate("mark record")
                v = int(command_line)
                mark.generate_random_marks(command_line)
                view.generated_message("mark record")
            except:
                view.incorrect_input_message("mark record")
        elif command == "search subjects_teachers records by name, classes, age":
            try:
                classes_range = view.ask_for_values_to_search("classes per semester range").split(", ")
                subject_name = view.ask_for_values_to_search("subject name")
                age_range = view.ask_for_values_to_search("teacher age range").split(", ")
                v1 = int(classes_range[0])
                v2 = int(classes_range[1])
                v3 = subject_name
                v4 = int(age_range[0])
                v5 = int(age_range[1])
                view.before_and_after_search("started")
                search.find_subjects_teachers_records_by_name_classes_age(classes_range, subject_name, age_range)
                view.before_and_after_search("ended")
```

```python
                view.before_and_after_search("ended")
            except:
                view.incorrect_input_message("search")
        elif command == "search schedule records by id, name, day":
            try:
                id_range = view.ask_for_values_to_search("id range").split(", ")
                teacher_name = view.ask_for_values_to_search("teacher's name")
                day = view.ask_for_values_to_search("day")
                v1 = int(id_range[0])
                v2 = int(id_range[1])
                v3 = teacher_name
                v4 = day
                view.before_and_after_search("started")
                search.find_schedule_records_by_id_name_day(id_range, teacher_name, day)
                view.before_and_after_search("ended")
            except:
                view.incorrect_input_message("search")
        elif command == "search marks records by name, mark, grade":
            try:
                mark_range = view.ask_for_values_to_search("mark range").split(", ")
                subject_name = view.ask_for_values_to_search("subject name")
                grade = view.ask_for_values_to_search("grade")
                v1 = int(mark_range[0])
                v2 = int(mark_range[1])
                v3 = subject_name
                v4 = int(grade)
                view.before_and_after_search("started")
                search.find_marks_records_by_name_mark_grade(mark_range, grade, subject_name)
                view.before_and_after_search("ended")
            except:
                view.incorrect_input_message("search")
        else:
            print("Unknown command, try again!")
```

Model.py:



471 lines (413 sloc) | 17.1 KB

```python
class Student:
    def __init__(self, connection):
        self.connection = connection


    def add_student(self, line):
        try:
            line_adding = line.split(", ")
            students = [(line_adding[0], line_adding[1], line_adding[2])]
            student_records = ", ".join(["%s"]*len(students))
            insert_query = (
                f"INSERT INTO students (name, age, grade) VALUES {student_records}"
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, students)
        except:
            print("Error: student was not added!")


    def update_student(self, line):
        try:
            line_editing = line.split(", ")
            update_student = f"""
            UPDATE
              students
            SET
              name = '{line_editing[1]}',
              age = '{line_editing[2]}',
              grade = '{line_editing[3]}'
            WHERE
              id = {line_editing[0]}
            """
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(update_student, line_editing)
        except:
            print("Error: student was not updated!")


    def delete_student(self, line_deleting):
        try:
```

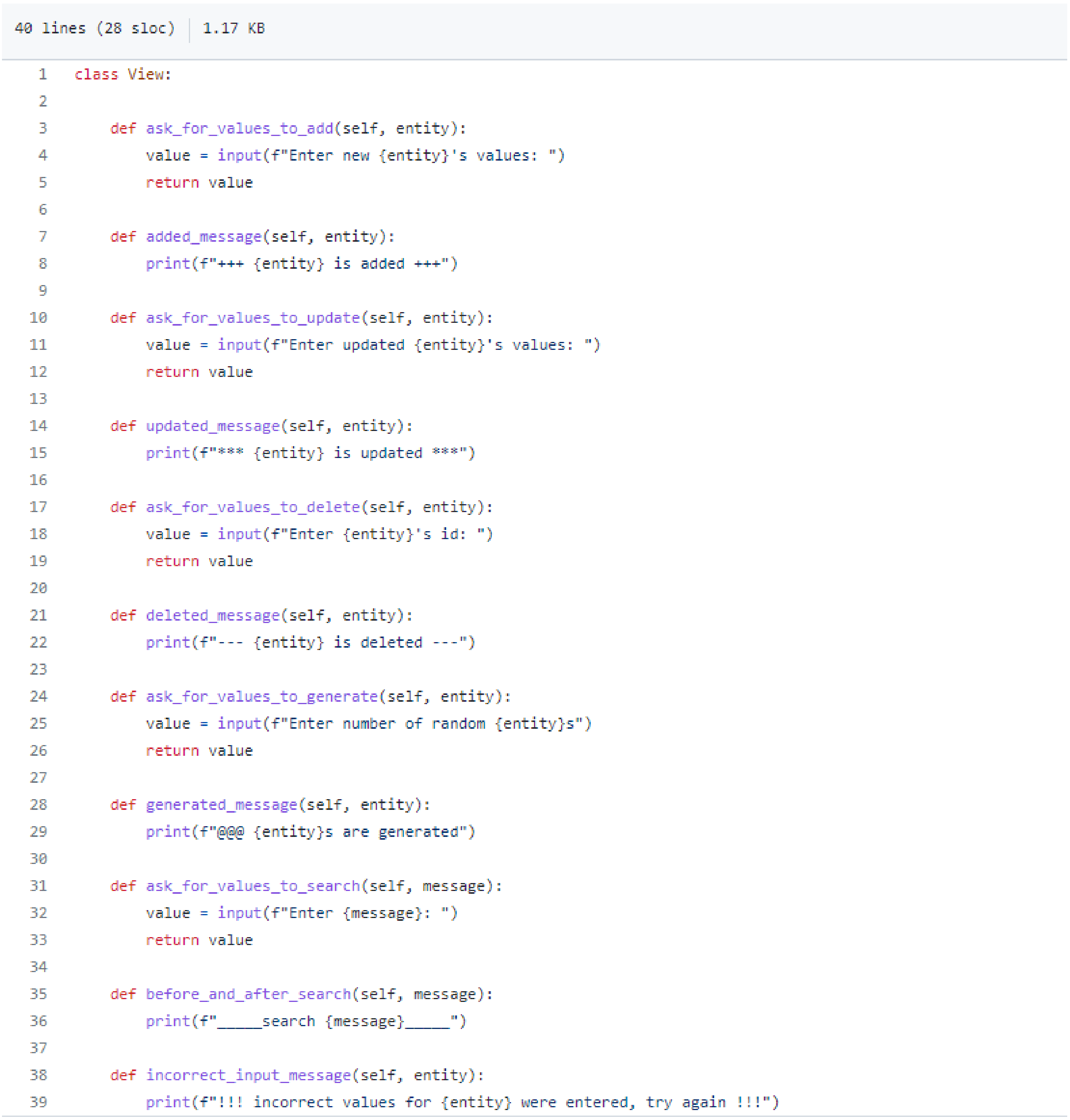
```python
        try:
            delete_student = f"DELETE FROM students WHERE id = '{line_deleting}'"
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(delete_student, line_deleting)
        except:
            print(f"Error: student with id = {line_deleting} does not exist!")


    def generate_random_students(self, line_adding):
        try:
            insert_query = (
                f"""INSERT INTO students(name, age, grade)
                SELECT
                chr(trunc(65+RANDOM()*25)::INT)||chr(trunc(65+RANDOM()*25)::INT) AS name,
                trunc(RANDOM() * 15 + 6)::INT AS age,
                trunc(RANDOM() * 10 + 2)::INT AS grade
                FROM GENERATE_SERIES(1, {line_adding}) seq;"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, line_adding)
        except:
            print("Error: student was not generated!")



class Teacher:
    def __init__(self, connection):
        self.connection = connection


    def add_teacher(self, line):
        try:
            line_adding = line.split(", ")
            teachers = [(line_adding[0], line_adding[1], line_adding[2])]
            teacher_records = ", ".join(["%s"] * len(teachers))
            insert_query = (
                f"INSERT INTO teachers (name, age, work_experience) VALUES {teacher_records}"
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, teachers)
        except:
            print("Error: teacher was not added!")
            print("Error: teacher was not added!")

    def update_teacher(self, line):
        try:
            line_editing = line.split(", ")
            update_teacher = f"""
            UPDATE
              teachers
            SET
              name = '{line_editing[1]}',
              age = '{line_editing[2]}',
              work_experience = '{line_editing[3]}'
            WHERE
              id = {line_editing[0]}
            """
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(update_teacher, line_editing)
        except:
            print("Error: teacher was not updated!")

    def delete_teacher(self, line_deleting):
        try:
            delete_teacher = f"DELETE FROM teachers WHERE id = '{line_deleting}'"
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(delete_teacher, line_deleting)
        except:
            print(f"Error: teacher with id = {line_deleting} does not exist!")

    def generate_random_teachers(self, line_adding):
        try:
            insert_query = (
                f"""INSERT INTO teachers(name, age, work_experience)
                SELECT
                chr(trunc(65+RANDOM()*25)::INT)||chr(trunc(65+RANDOM()*25)::INT) AS name,
                trunc(RANDOM() * 10 + 6)::INT AS age,
                trunc(RANDOM() * 10 + 10)::INT AS work_experience
                FROM GENERATE_SERIES(1, {line_adding}) seq;"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, line_adding)
        except:
            print("Error: teacher was not generated!")
```

```python
            print("Error: teacher was not generated!")




class Subject:
    def __init__(self, connection):
        self.connection = connection


    def add_subject(self, line):
        try:
            line_adding = line.split(", ")
            subjects = [(line_adding[0], line_adding[1])]
            subject_records = ", ".join(["%s"] * len(subjects))
            insert_query = (
                f"INSERT INTO teachers (name, classes_per_semester) VALUES {subject_records}"
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, subjects)
        except:
            print("Error: subject was not added!")

    def update_subject(self, line):
        try:
            line_editing = line.split(", ")
            update_subject = f"""
            UPDATE
              subjects
            SET
              name = '{line_editing[1]}',
              classes_per_semester = '{line_editing[2]}'
            WHERE
              id = {line_editing[0]}
            """
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(update_subject, line_editing)
        except:
            print("Error: subject was not updated!")

    def delete_subject(self, line_deleting):
        try:
            delete_subject = f"DELETE FROM subjects WHERE id = '{line_deleting}'"
            delete_subject = f"DELETE FROM subjects WHERE id = '{line_deleting}'"
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(delete_subject, line_deleting)
        except:
            print(f"Error: subject with id = {line_deleting} does not exist!")

    def generate_random_subjects(self, line_adding):
        try:
            insert_query = (
                f"""INSERT INTO subjects(name, classes_per_semester)
                SELECT
                chr(trunc(65+RANDOM()*25)::INT)||chr(trunc(65+RANDOM()*25)::INT) AS name,
                trunc(RANDOM() * 10 + 6)::INT AS classes_per_semester
                FROM GENERATE_SERIES(1, {line_adding}) seq;"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, line_adding)
        except:
            print("Error: subject was not generated!")



class Subject_Teacher:
    def __init__(self, connection):
        self.connection = connection


    def add_subjects_teachers_record(self, line):
        try:
            line_adding = line.split(", ")
            records = [(line_adding[0], line_adding[1])]
            subjects_teachers_records = ", ".join(["%s"] * len(records))
            insert_query = (
                f"INSERT INTO subjects_teachers (student_id, teacher_id) VALUES {subjects_teachers_records}"
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, records)
        except:
            print("Error: record was not added!")

    def update_subjects_teachers_record(self, line):
        try:
```

```
        try:
            line_editing = line.split(", ")
            update_record = f"""
            UPDATE
              subjects_teachers
            SET
              subject_id = '{line_editing[1]}',
              teacher_id = '{line_editing[2]}'
            WHERE
              id = {line_editing[0]}
            """
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(update_record, line_editing)
        except:
            print("Error: record was not updated!")

    def delete_subjects_teachers_record(self, line_deleting):
        try:
            delete_record = f"DELETE FROM subjects_teachers WHERE id = '{line_deleting}'"
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(delete_record, line_deleting)
        except:
            print(f"Error: record with id = {line_deleting} does not exist!")

    def generate_random_subjects_teachers_records(self, line_adding):
        try:
            insert_query = (
                f"""INSERT INTO subjects_teachers(subject_id, teacher_id)
                SELECT
                trunc(RANDOM() * 10 + 6)::INT AS subject_id,
                trunc(RANDOM() * 10 + 3)::INT AS teacher_id
                FROM GENERATE_SERIES(1, {line_adding}) seq;"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, line_adding)
        except:
            print("Error: record was not generated!")




class Schedule:
```

```python
class Schedule:
    def __init__(self, connection):
        self.connection = connection


    def add_schedule_record(self, line):
        try:
            line_adding = line.split(", ")
            record = [(line_adding[0], line_adding[1], line_adding[2], line_adding[3])]
            schedule_records = ", ".join(["%s"] * len(record))
            insert_query = (
                f"INSERT INTO scedule (day, time, subject_teacher_id, student_id) VALUES {schedule_records}"
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, record)
        except:
            print("Error: record was not added!")

    def update_schedule_record(self, line):
        try:
            line_editing = line.split(", ")
            update_record = f"""
            UPDATE
              scedule
            SET
              day = '{line_editing[1]}',
              time = '{line_editing[2]}',
              subject_teacher_id = '{line_editing[3]}',
              student_id = '{line_editing[4]}'
            WHERE
              id = {line_editing[0]}
            """
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(update_record, line_editing)
        except:
            print("Error: record was not updated!")

    def delete_schedule_record(self, line_deleting):
        try:
            delete_record = f"DELETE FROM scedule WHERE id = '{line_deleting}'"
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(delete_record, line_deleting)
            cursor.execute(delete_record, line_deleting)
        except:
            print(f"Error: record with id = {line_deleting} does not exist!")

    def generate_random_schedule_records(self, line_adding):
        try:
            insert_query = (
                f"""INSERT INTO scedule(day, time, subject_teacher_id, student_id)
                SELECT
                CASE trunc(RANDOM() * 10)::INT
                 WHEN 0 THEN 'monday'
                 WHEN 1 THEN 'tuesday'
                 WHEN 2 THEN 'wednesday'
                 WHEN 3 THEN 'thursday'
                 WHEN 4 THEN 'friday'
                 ELSE 'saturday'
                END AS day,
                trunc(RANDOM() * 10 + 5)::INT||':'||trunc(RANDOM() * 10 + 5)::INT AS time,
                trunc(RANDOM() * 2 + 1)::INT AS subject_teacher_id,
                trunc(RANDOM() * 2 + 1)::INT AS student_id
                FROM GENERATE_SERIES(1, {line_adding}) seq;"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(insert_query, line_adding)
        except:
            print("Error: record was not generated!")




class Mark:
    def __init__(self, connection):
        self.connection = connection

    def add_mark(self, line):
        try:
            line_adding = line.split(", ")
            marks = [(line_adding[0], line_adding[1], line_adding[2])]
            mark_records = ", ".join(["%s"] * len(marks))
            insert_query = (
                f"INSERT INTO marks (student_id, subject_teacher_id, mark) VALUES {mark_records}"
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor = self.connection.cursor()
            cursor.execute(insert_query, marks)
        except:
            print("Error: mark was not added!")

    def update_mark(self, line):
        try:
            line_editing = line.split(", ")
            update_mark = f"""
               UPDATE
                 marks
               SET
                 name = '{line_editing[1]}',
                 subject_teacher_id = '{line_editing[2]}',
                 mark = '{line_editing[3]}'
               WHERE
                 id = {line_editing[0]}
               """
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(update_mark, line_editing)
        except:
            print("Error: mark was not updated!")

    def delete_mark(self, line_deleting):
        try:
            delete_mark = f"DELETE FROM marks WHERE id = '{line_deleting}'"
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(delete_mark, line_deleting)
        except:
            print(f"Error: mark with id = {line_deleting} does not exist!")

    def generate_random_marks(self, line_adding):
        try:
            insert_query = (
                f"""INSERT INTO marks(student_id, subject_teacher_id, mark)
                SELECT
                trunc(RANDOM() * 2 + 1)::INT AS student_id,
                trunc(RANDOM() * 2 + 1)::INT AS subject_teacher_id,
                trunc(RANDOM() * 10 + 5)::INT AS mark
                FROM GENERATE_SERIES(1, {line_adding}) seq;"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor = self.connection.cursor()
            cursor.execute(insert_query, line_adding)
        except:
            print("Error: mark was not generated!")




class Search:
    def __init__(self, connection):
        self.connection = connection


    def find_subjects_teachers_records_by_name_classes_age(self, classes_range, subject_name, age_range):
        try:
            find_by_value_query = (
                f"""SELECT DISTINCT sub.name, sub.classes_per_semester, teach.name, teach.age
                    FROM subjects sub, teachers teach, subjects_teachers st
                    WHERE
                        st.subject_id = sub.id
                        AND sub.name = '{subject_name}'
                        AND sub.classes_per_semester > {classes_range[0]}
                        AND sub.classes_per_semester < {classes_range[1]}
                        AND teach.age > {age_range[0]}
                        AND teach.age < {age_range[1]}"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(find_by_value_query)
            for line in cursor.fetchall():
                print(line)
        except:
            print("Error: records were not found!")

    def find_schedule_records_by_id_name_day(self, id_range, teacher_name, day):
        try:
            find_by_value_query = (
                f"""SELECT DISTINCT sc.id, sc.day, sc.time, sub.name, teach.name, stud.name
                FROM scedule sc, subjects sub, teachers teach, students stud, subjects_teachers st
                WHERE
                    sc.id > {id_range[0]}
                    AND sc.id < {id_range[1]}
                    AND st.id = sc.subject_teacher_id
                    AND sub.id = st.subject_id
                    AND teach.id = st.teacher_id
                    AND teach.id = st.teacher_id
                    AND stud.id = sc.student_id
                    AND teach.name LIKE '%{teacher_name}%'
                    AND sc.day = '{day}'"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(find_by_value_query)
            for line in cursor.fetchall():
                print(line)
        except:
            print("Error: records were not found!")

    def find_marks_records_by_name_mark_grade(self, mark_range, grade, subject_name):
        try:
            find_by_value_query = (
                f"""SELECT DISTINCT ma.mark, stud.name, stud.grade, sub.name, teach.name
                    FROM subjects sub, teachers teach, subjects_teachers st, marks ma, students stud
                    WHERE
                        ma.mark >= {mark_range[0]}
                        AND ma.mark <= {mark_range[1]}
                        AND stud.id = ma.student_id
                        AND st.id = ma.subject_teacher_id
                        AND teach.id = st.teacher_id
                        AND stud.grade = '{grade}'
                        AND sub.name LIKE '%{subject_name}%'"""
            )
            self.connection.autocommit = True
            cursor = self.connection.cursor()
            cursor.execute(find_by_value_query)
            for line in cursor.fetchall():
                print(line)
        except:
            print("Error: records were not found!")
```

View.py:



40 lines (28 sloc) | 1.17 KB

```python
class View:

    def ask_for_values_to_add(self, entity):
        value = input(f"Enter new {entity}'s values: ")
        return value

    def added_message(self, entity):
        print(f"+++ {entity} is added +++")

    def ask_for_values_to_update(self, entity):
        value = input(f"Enter updated {entity}'s values: ")
        return value

    def updated_message(self, entity):
        print(f"*** {entity} is updated ***")

    def ask_for_values_to_delete(self, entity):
        value = input(f"Enter {entity}'s id: ")
        return value

    def deleted_message(self, entity):
        print(f"--- {entity} is deleted ---")

    def ask_for_values_to_generate(self, entity):
        value = input(f"Enter number of random {entity}s")
        return value

    def generated_message(self, entity):
        print(f"@@@ {entity}s are generated")

    def ask_for_values_to_search(self, message):
        value = input(f"Enter {message}: ")
        return value

    def before_and_after_search(self, message):
        print(f"_____search {message}_____")

    def incorrect_input_message(self, entity):
        print(f"!!! incorrect values for {entity} were entered, try again !!!")
```